The Report Committee for Qi Chen
certifies that this is the approved version of the following report:

Machine Learning Phases in Statistical Physics

APPROVED BY

SUPERVISING COMMITTEE:

Timothy H. Keitt, Supervisor

Gregory A. Fiete

Machine Learning Phases in Statistical Physics

by

Qi Chen

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN STATISTICS

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2017

Dedicated to all authors, who took the time to write their knowledge down.

Acknowledgments

I wish to thank Dr. Tim Keitt for supervising this thesis.

I'd also like to thank Dr. Gregory A. Fiete for supervising my Ph.D work and being in support of my concurrent M.S. in statistics.

I acknowledge Juan Felipe Carrasquilla Alvarez for his very helpful discussion closely related to this report.

Thanks to Vicki, Keller and all my colleagues in Department of Statistics and Department of Physics for their help.

Lastly, my sincere love for my parents.

Machine Learning Phases in Statistical Physics

Qi Chen, M. S. Stat.
The University of Texas at Austin, 2017

Supervisor: Timothy H. Keitt

ABSTRACT Conventionally, the study of phases in statistical mechanics is performed with the help of random sampling tools. Among the most powerful are Monte Carlo simulations consisting of a stochastic importance sampling over state space and evaluation of estimators for physical quantities. The ability of modern machine learning techniques to classify, identify, or interpret massive data sets provides a complementary paradigm to the above approach to analyze the exponentially large number of states in statistical physics. In this report, it is demonstrated by application on Ising-type models that deep learning has potential wide applications in solving many-body statistical physics problems. In application of supervised learning, we showed that the feed-forward neural network can identify phases and phase transitions in the ferromagnetic Ising model and the convolutional neural network (CNN) is extremely powerful in classifying $T = 0$ and $T = \infty$ phases in the Ising gauge model; In application of unsupervised learning, we illustrated that a deep auto-encoder constructed by stacked restricted Boltzmann machines (RBM)

is closely related to the renormalization group (RG) method well understood in modern physics and our reconstruction of Ising spin configurations in the ferromagnetic Ising model is similar to the hand-written digits reconstruction.

Table of Contents

Acknowledgments **v**

Abstract **vi**

List of Tables **x**

List of Figures **xi**

Chapter 1. Introduction **1**
1.1 Statistical Physics 1
1.2 Machine Learning 3
1.3 Deep Learning in statistical physics 4

Chapter 2. Monte Carlo methods in statistical physics **7**
2.1 Importance sampling through Markov chains 7
2.2 Monte Carlo for the Ising model 9
2.3 Monte Carlo for the Ising lattice gauge model 11

Chapter 3. Deep learning models **16**
3.1 Feed-forward neural network 16
3.1.1 Logistic regression 17
3.1.2 Multi-class classification 18
3.1.3 Multi-layer Perceptron 18
3.1.4 Softmax and cost function 19
3.1.5 Activation functions 19
3.1.6 Backpropagation 20
3.2 Convolutional neural network 21
3.2.1 The Convolution Operation 22
3.2.2 Motivation 22

3.2.3 Pooling 23
3.2.4 Full CNN Model 23
3.3 Restricted Boltzmann Machine 24
3.3.1 Bernoulli Restricted Boltzmann machines 25
3.3.2 Maximum Likelihood Learning 25
3.3.3 Contrastive divergence 26
3.4 Deep auto-encoder 27

Chapter 4. Data-preparation and model specification 33
4.1 Ferromagnetic Ising model 33
4.1.1 Data 33
4.1.2 Model 34
4.2 Ising gauge model 35
4.2.1 Data 35
4.2.2 Model 36

Chapter 5. Results 37
5.1 Feed-forward ANN on ferromagnetic Ising model 37
5.2 Deep auto-encoder on ferromagnetic Ising model 39
5.3 CNN on Ising gauge model 42

Chapter 6. Conclusion and discussion 45

Appendix 47

Appendix 1. Sample code 48
1.1 Keras implementation of feed-forward ANN in ferromagnetic Ising model 48
1.2 Keras implementation of CNN in Ising gauge model 52

Bibliography 60

Vita 63

List of Tables

4.1 Sample data frame for $L = 20$ Ising model. 34

5.1 Training and test accuracy for ferromagnetic Ising model. . . . 37

List of Figures

2.1 Ferromagnetic Ising model on the square lattice: Each spin is defined on the site of the lattice and takes binary values from $\{-1, 1\}$. 10

2.2 Ising gauge model on the square lattice: Each spin is defined on the link of the lattice and takes binary values from $\{-1, 1\}$. 12

2.3 Non-local flux insertion along x or y direction. 14

3.1 The architecture of a feed-forward artificial neural network with a single hidden layer. The output probabilities of each neuron in the hidden layer are served as inputs for the output layer. (This figure was clipped from online course materials at coursera.com) 30

3.2 The architecture of CNN. The input is a colored picture with a 2d grid of pixels. Two convolution+pooling layers are used to convert pixels into 1d input data for the fully connected layer as in the feed-forward neural network[1]. 31

3.3 The architecture of restricted Boltzmann machine: the hidden units on the top are connected to the visible units at the bottom. There is no internal connections between units in the hidden layer[2]. 31

3.4 The architectures of auto-encoders[3]: (a) Auto-encoder with a single hidden layer and the sigmoid activation function. (b) Deep auto-encoder with stacked RBMs as encoding layers and the decoding layers have the reversed dimensions. 32

5.1 The output layer of our trained feed-forward neural network model averaged over a test set as a function of T/J for the square-lattice ferromagnetic Ising model of size $L = 10, 20, 30, 40, 60$. 38

5.2 The plot for finite size scaling of the critical temperature $T_c/J \sim 1/L$. The red line is the fitted result. 39

5.3 Visualization of effective receptive fields for (a) 100 spins in the middle layer, (b) 25 spins in the top layer. Each 40 by 40 pixel image depicts the effective receptive field of one of the spins in the hidden units. 40

5.4 Loss per data point in the fine tuning stage as a function of number of epochs trained for the deep auto-encoder. The green curve corresponds to random initialization in the model parameter. The blue curve is initialized with pre-trained model parameters. 42

5.5 Reconstructed spin configurations (lower) compared to the corresponding raw configurations (upper). 43

5.6 The spin flip transformations on the links in x direction every m plaquettes for $L = 20$, where $m = 2$ for the upper row and $m = 4$ for the lower row. From left to right: spin configurations before the transformation, spin configurations after the transformation, spins changed under the transformation indicated by the white spots. 44

Chapter 1

Introduction

In this chapter, we introduce the basic knowledge about statistical physics[4] and machine learning[5], which motivates our work on specific models. In 1.1 we present the basic knowledge of statistical physics. In 1.2 we introduce the basics of machine learning. In 1.3, we address our motivation for the work on deep learning application in statistical physics and the organization of following chapters.

1.1 Statistical Physics

Statistical physics is a branch of physics that uses methods of probability theory and statistics, and particularly the mathematical tools and numerical techniques for dealing with large populations and approximations, in solving physical problems. It can describe a wide variety of fields with an inherently stochastic nature.

In particular, statistical mechanics develops the phenomenological results of thermodynamics from a probabilistic examination of the underlying microscopic systems. It provides a framework for relating the microscopic properties of individual atoms and molecules to the macroscopic or bulk prop-

erties of materials that can be observed in every day life, therefore explaining thermodynamics as a natural result of statistics, classical mechanics, and quantum mechanics at the microscopic level.

One of the most important equations in Statistical mechanics is the definition of the partition function Z, which is essentially a weighted sum of all possible states q available to a system:

$$Z = \sum_q exp(-E(q)/k_B T), \tag{1.1}$$

where k_B is the Boltzmann constant, T is temperature and $E(q)$ is energy of state q.The probability of a given state q is usually decided by the energy of state q and also the temperature:

$$P(q) = \frac{exp(-E(q)/k_B T)}{Z}. \tag{1.2}$$

The macroscopic observable $\mathcal{O}$' s are weighted averages of microscopic degrees of freedoms associated with q's:

$$< \mathcal{O} >= \sum_q P(q)\mathcal{O}(q). \tag{1.3}$$

In classical systems, phases of matters are characterized by so called order parameters–the weighted averages of particular choices of physical observables, e.g. magnetic moment. Phase transitions happen when there is an abrupt change in the order parameter as a function of temperature. Although some phase transition problems in statistical physics can be solved analytically using approximations and expansions, most current research utilizes the large

processing power of modern computers to simulate or approximate solutions. A common approach to statistical problems is to use a Monte Carlo simulation to yield insight into the dynamics of a complex system.

1.2 Machine Learning

Machine learning lives in the cross field between statistics and computer science, exploring the study and construction of algorithms that can learn from and make predictions on data. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders or malicious insiders working towards a data breach, optical character recognition (OCR), learning to rank, and computer vision.

Machine learning can be supervised or unsupervised depending on whether the desired outputs are given or not. The goal of supervised learning is to learn a general rule that maps inputs to outputs, whereas unsupervised learning can be used to discover hidden patterns or as a means towards an end such as feature selection. Within the field of data science, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction.

In particular, deep learning is the application to learning tasks of artificial neural networks (ANNs) that contain more than one hidden layer. It is a class of machine learning algorithms that:

1. use a cascade of many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. The algorithms may be supervised or unsupervised and applications include pattern analysis (unsupervised) and classification (supervised).

2. are based on the (unsupervised) learning of multiple levels of features or representations of the data. Higher level features are derived from lower level features to form a hierarchical representation.

3. are part of the broader machine learning field of learning representations of data.

4. learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

Deep learning has been a rising part of machine learning methods based on learning data representations, as opposed to task specific algorithms. The architectures of deep learning includes deep neural networks, deep belief networks and recurrent neural networks that having been widely applied to fields including computer vision, speech recognition, natural language processing, etc.

1.3 Deep Learning in statistical physics

Conventionally, the study of phases in statistical mechanics is performed with the help of tools that have been carefully designed to elucidate

the underlying configurations of various states. Among the most powerful are Monte Carlo[15] simulations consisting of a stochastic importance sampling over state space, and the evaluation of estimators for physical quantities calculated from samples. Machine learning, already explored as a tool in condensed-matter research[7, 6, 13, 12, 9, 16, 14], provides physicists a complementary paradigm in analyzing exponentially large data sets embodied in the state space of statistical physics models. In this report, we closely followed the work by Juan[7], Pankaj[14] and applied deep learning models in studying phases of matter based on data sets sampled by Monte Carlo simulations. We first demonstrated the power of supervised feed-forward artificial neural network (ANN) in the classification of phases in the prototypical example of the square-lattice ferromagnetic Ising model. Then we turned to a more complicated example of Ising gauge model and examined the ability of convolutional neural network (CNN) to detect phase transitions through the local constraints in the flux through a plaquette. Finally, we showcased the power of deep autoencoders consisting of multiple layers of restricted Boltzmann machine (RBM) in reconstructing spin configurations in the microscopic states of ferromagnetic Ising model close to the critical point of phase transition.

This report is organized as follows. In chapter 2, we introduce the Markov chain Monte Carlo sampling method with the Metropolis algorithm and its application in two prototypical statistical physics models: ferromagnetic Ising model and Ising gauge model. In Chapter 3, feed-forward ANN and CNN are reviewed and specified in our application. In addition, deep

auto-encoders and their building blocks–RBMs are introduced as an ANN for unsupervised learning. We point out the intimate relationship between renormalization group methods and RBM and specified the details of stacked RBMs in encoding and decoding spin configurations in ferromagnetic Ising models. In Chapter 4, we elaborate our data preparation process and specify the hyperparameters in our models. In Chapter 5 we report our training and testing results for both supervised and unsupervised learning. Conclusions and further discussions are made in Chapter 6.

Chapter 2

Monte Carlo methods in statistical physics

The Monte Carlo method is a traditional numerical tool for simulating classical many-particle systems by introducing artificial dynamics based on 'random' numbers. For our purposes, we will use Monte Carlo sampling to generate spin configurations data for the Ising type models. We will first introduce the Metropolis algorithm in the Markov chain Monte Carlo method and then apply it to both the ferromagnetic Ising model and the Ising gauge model.

2.1 Importance sampling through Markov chains

In this section, we explain the importance sampling method for classical many-particle systems in the canonical (N, V, T) ensemble[17]. When calculating averages in the (N, V, T) ensemble, the configuration X should be weighted according to the Boltzmann factor

$$\rho(X) \propto \exp[-\beta E(X)], \beta = \frac{1}{k_B T} \tag{2.1}$$

This suggests that we apply Monte Carlo integration with importance sampling. However, as the number of configurations with a particular energy increases exponentially, most of the randomly constructed states have very high

energy-hence for finite temperature they will be accepted with a vanishingly small probability and we spend most of our time generating configurations which are then rejected. As a result, importance sampling becomes very inefficient. To improve efficiency in sampling, the configurations are constructed through a Markov chain, in which each new configuration X' is generated with a probability distribution depending on the previous configuration X given by $T(X \to X')$. Then the probability of having a sequence of objects X_i becomes

$$P_N(X_1, X_2, \ldots, X_N) = P_1(X_1)\, T(X_1 \to X_2)\, T(X_2 \to X_3) \ldots T(X_{N-1} \to X_N). \tag{2.2}$$

In our problem, we want to generate a Markov chain of system configurations such that they have a distribution proportional to $\exp[-\beta E(X)]$. In other words, we want to find a transition probability $T(X \to X')$ which leads to a given stationary density $\rho(X)$. This is achieved by the Metropolis Monte Carlo method, in which $T(X \to X')$ is constructed as

$$T(X \to X') = \omega_{XX'} A_{XX'}, \tag{2.3}$$

where $\omega_{XX'}$ is the proposal distribution and $A_{XX'}$ is the acceptance probability. As $\rho(X)$ is the stationary distribution, we want

$$T(X \to X')\, \rho(X) = T(X' \to X)\, \rho(X'). \tag{2.4}$$

By inserting Eq. (3) into Eq. (4), one finds

$$\omega_{XX'} A_{XX'} \rho(X) = \omega_{X'X} A_{X'X} \rho(X') \Rightarrow \frac{A_{XX'}}{A_{X'X}} = \frac{\omega_{X'X} \rho(X')}{\omega_{XX'} \rho(X)}. \tag{2.5}$$

In the Metropolis procedure, the acceptance criterion for an attempted move $X \to X'$ is given by

$$A_{XX'} = \min\left\{1, \frac{\omega_{X'X}\rho\left(X'\right)}{\omega_{XX'}\rho(X)}\right\} \tag{2.6}$$

2.2 Monte Carlo for the Ising model

The Hamiltonian (i.e. energy) of the ferromagnetic Ising model defined on the $L \times L$ square lattice (Fig. 2.1) is given as:

$$H = -J \sum_{<ij>} S_i^z S_j^z, \tag{2.7}$$

where the Ising spins S_i^z live on the sites of a two-dimensional square lattice and takes binary values from $\{-1, 1\}$. In total, we have $N = L \times L$ sites and 2^N states. This is a textbook model in classical statistical mechanics which describes the phase transition from ferromagnetic (FM) phase ($| < S_i^z > | \approx 1$) at low temperatures to paramagnetic (PM) phase ($| < S_i^z > | \approx 0$) at high temperatures. In order to formulate the Metropolis procedure, we must make a choice for $\omega_{XX'}$. For the two-dimensional Ising model on an $L \times L$ square lattice, we can take

$$\omega_{XX'} = \begin{cases} 1/L^2 & \text{if } X \text{ and } X' \text{ differ by one spin} \\ 0 & \text{otherwise} \end{cases}. \tag{2.8}$$

In order to generate a trial configuration, one can select a spin at random and the trial configuration is the present configuration with the selected spin turned over. The energy difference is given by

$$\Delta E\left(X \to X'\right) = E\left(X'\right) - E(X). \tag{2.9}$$

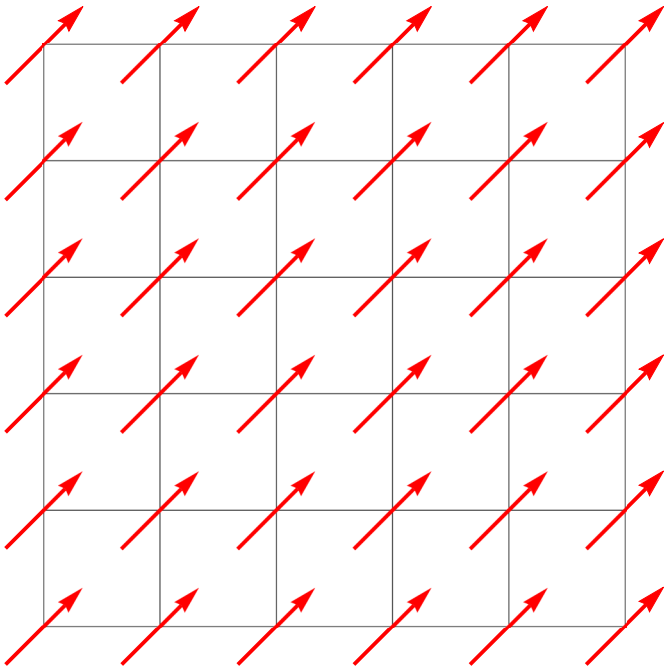


Figure 2.1: Ferromagnetic Ising model on the square lattice: Each spin is defined on the site of the lattice and takes binary values from $\{-1, 1\}$.

On the square lattice in two dimension, it is clear that the energy difference depends only on the number of neighbors which have the same spin as the selected spin in the old configuration - between 0 and 4. the energy differences are

$$\begin{pmatrix} \Delta E\,(X \to X') & \#\text{ of neighbors with the same spin} \\ -8J & 0 \\ -4J & 1 \\ 0 & 2 \\ 4J & 3 \\ 8J & 4 \end{pmatrix}. \tag{2.10}$$

The trial state is accepted with probability:

$$A_{XX'} = \begin{cases} \exp\left(-\beta \Delta E\left(X \to X'\right)\right) & \text{if } \Delta E\left(X \to X'\right) > 0 \\ 1 & \text{else} \end{cases} \tag{2.11}$$

Since $\exp\left(-\beta \Delta E\left(X \to X'\right)\right)$ can assume only five different values, it is advisable to store these in an array in order to avoid calculating the exponential function over and over again. Notice that the average number of steps between two updates of the same spin is equal to L^2. Therefore, the Monte Carlo steps per spin(MCS) is equal to L^2 trials. (Remark: The initial configuration will be chosen either random or as FM (all spins taking either $+1$ or -1).) The specific heat is calculated according to

$$\begin{aligned} C_V &= \frac{1}{k_B T^2} \frac{\partial^2 \mathrm{Log}[Z]}{\partial \beta^2} \\ &= \frac{1}{k_B T^2} \left[\frac{\sum_X e^{-\beta H(X)} H^2(X)}{\sum_X e^{-\beta H(X)}} - \left(\frac{\sum_X e^{-\beta H(X)} H(X)}{\sum_X e^{-\beta H(X)}} \right)^2 \right] \\ &= \frac{1}{k_B T^2} \left[\left\langle E^2 \right\rangle_{\mathrm{NVT}} - \langle E \rangle^2_{\mathrm{NVT}} \right]. \end{aligned} \tag{2.12}$$

Similarly, the magnetic susceptibility can be written as

$$\chi = \beta \left[\left\langle M^2 \right\rangle - \langle M \rangle^2 \right]. \tag{2.13}$$

2.3 Monte Carlo for the Ising lattice gauge model

The Hamiltonian for the Ising gauge model[8] (Fig. 2.2) is

$$H = -J \sum_p \prod_{i \in p} \sigma_i^z, \tag{2.14}$$

where the Ising spins live on the bonds of a two-dimensional square lattice with plaquettes p and takes binary values from $\{-1, 1\}$. This model has a

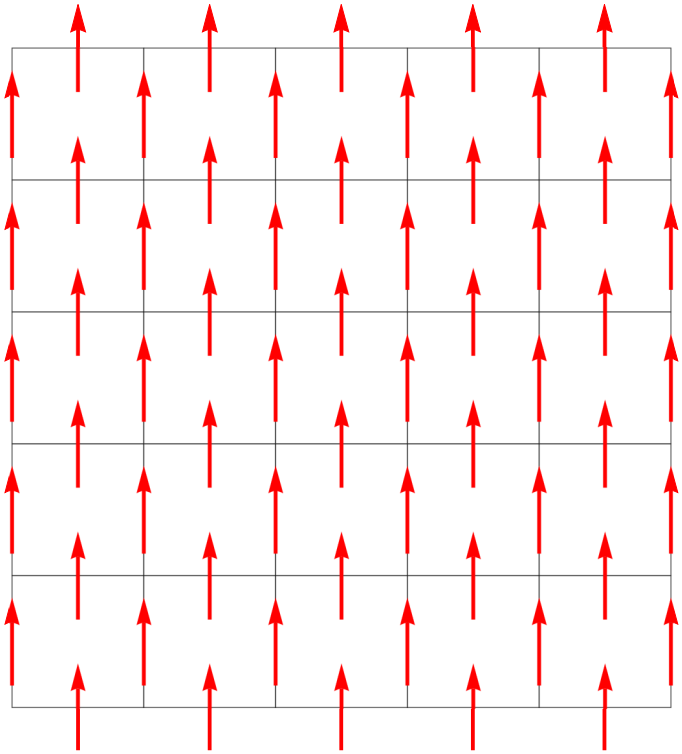


Figure 2.2: Ising gauge model on the square lattice: Each spin is defined on the link of the lattice and takes binary values from $\{-1, 1\}$.

gauge symmetry, i.e. the Hamiltonian is invariant under the following transformation:

$$\tilde{\sigma}^z_{ij} = w_i \sigma^z_{ij} w_j \tag{2.15}$$

where $w_i = \pm 1$. We can count the number of states in the Ising gauge theory on a finite square lattice with periodic boundary condition as follows: If the lattice has N sites, then it has $2N$ nearest-neighbor links. Thus there are 2^{2N} different σ^z_{ij} configurations. On the other hand, there are 2^N different gauge transformations, which induces $2^N / 2$ different gauge-equivalent classes. If we

choose to label those classes by plaquette values, one naively expects that different plaquette configurations will provide 2^N different labels. However, the plaquette values are not independent under periodic boundary condition. They satisfy

$$\prod_p \prod_{i \in p} \sigma_i^z = 1 \tag{2.16}$$

So they only provide $2^N \big/ 2$ states. As a result, each plaquette configuration corresponds to four different states. To see this (Fig. 2.3), we consider the dashed line along x or y direction where all vertical links are flipped, which gives rise to the same plaquette configurations but they are gauge inequivalent because they correspond to inserting a different Z_2 flux through the two holes in the torus. Therefore, $2^N \big/ 2$ different flux configurations plus the four-fold degeneracy allow us to recover 2×2^N states[8]. Similar to FM Ising model, in order to formulate the Metropolis procedure, we must make a choice for $\omega_{XX'}$. Notice that in two dimension, there are $2 \times N$ bonds for N sites. Thus

$$\omega_{XX'} = \begin{cases} 1/2L^2 & \text{if } X \text{ and } X' \text{ differ by one spin on one link} \\ 0 & \text{else} \end{cases}. \tag{2.17}$$

The step of generating a trial configuration is again to select a spin at random, and the trial configuration is the present configuration with the selected spin turned over. The energy difference is given by

$$\Delta E\left(X \to X'\right) = E\left(X'\right) - E(X) \tag{2.18}$$

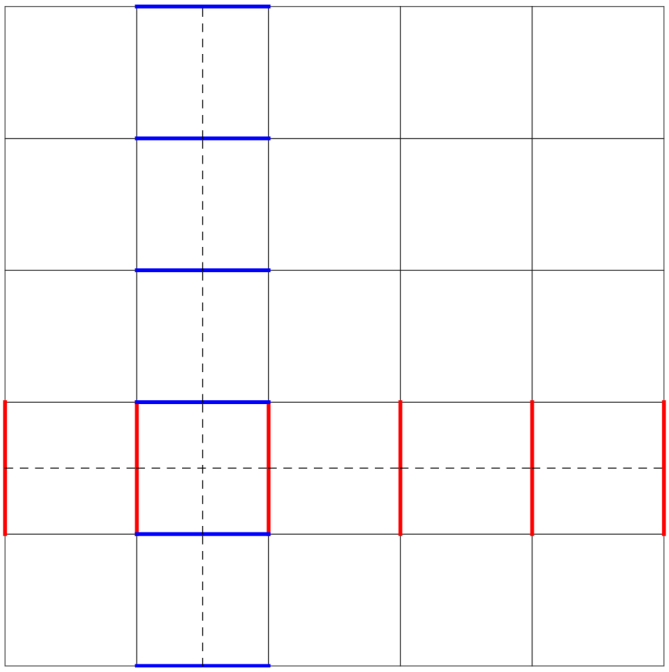


Figure 2.3: Non-local flux insertion along x or y direction.

On the square lattice in two dimension, it is clear that the energy difference depends only on the neighboring plaquettes flux:

$$\begin{pmatrix} \Delta E\left(X \to X'\right) & \text{neighboring plaquettes flux} \\ -4J & (-1,-1) \\ 0 & (-1,1) \text{ or } (1,-1) \\ 4J & (1,1) \end{pmatrix}. \tag{2.19}$$

The trial state is accepted with probability:

$$A_{XX'} = \begin{cases} \exp\left(-\beta \Delta E\left(X \to X'\right)\right) & \text{if } \Delta E\left(X \to X'\right) > 0. \\ 1 & \text{otherwise.} \end{cases} \tag{2.20}$$

Since $\exp\left[-\beta \Delta E\left(X \to X'\right)\right]$ can assume only 3 different values, it is advisable to store these in an array in order to avoid calculating the exponential function over and over again. The average number of steps between two updates of the

same spin is equal to $2L^2$. Therefore, the Monte Carlo steps per spin (MCS) is equal to $2L^2$ trials. However, the above procedure will not work for $T \to 0$ corresponding to the strong coupling limit ($J \to \infty$) as a single spin flip will cost infinite energy. To keep the chain moving, the most natural choice is to given the system the possibility of performing both single spin flips and local gauge transformations[18]. The relative number of both types of excitations may be chosen freely. With this in mind we choose $\omega_{XX'}$ as:

$$\omega_{XX'} = \begin{cases} \frac{s}{2L^2} & X \overset{\text{local}}{\to} X' \\ \frac{2(1-s)}{2L^2} & X \overset{\text{gauge}}{\to} X' \\ 0 & \text{otherwise} \end{cases} . \tag{2.21}$$

Our choice is based on the fact that there are L^2 gauge updates and $2L^2$ single spin flips.

Chapter 3

Deep learning models

In this chapter, we introduce the essential deep learning tools applied to our models. In 3.1, feed-forward ANN is introduced. In 3.2, we address the practical aspects of the convolutional neural network without getting bogged down in the theoretical details. In 3.3, the restricted Boltzmann machine is introduced with contrastive divergence as an approximation in gradient descent. In 3.4, we illustrate the structure of the deep auto-encoder based on stacked RBMs and point out the necessity of pre-training.

3.1 Feed-forward neural network

Feedforward ANN (Fig. 3.1) has been widely applied in multi-class classification problems, which is based upon multiple hidden layers with logistic hidden units. We first introduce the logistic regression in binary classification, then we generalize it to multi-class classification problems. With this building block, one can construct multi-layer perceptrons in a deep neural network and the predicted probabilities in the output layer are given by the softmax function. Alternatively, different activation functions can be used to construct hidden units. To achieve the minimization of the cost function, we

introduce the backpropagation method in evaluating the gradient descent of a deep neural network.

3.1.1 Logistic regression

The basic model for binary classification is the logistic regression based on the sigmoid function:

$$h_\theta(x) = g(x\theta) = \frac{1}{1+e^{-x\theta}}, \tag{3.1}$$

where the input variable x is an n-dimensional vector and θ is the weight vector for binary classification:

$$x = (x_1, x_2, \ldots, x_n), \theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{pmatrix}, \tag{3.2}$$

The learning goal is to maximize the log likelihood function:

$$J(\theta) = \sum_{i=1}^{m} \left[y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1-y^{(i)}\right) \log\left(1-h_\theta\left(x^{(i)}\right)\right) \right], \tag{3.3}$$

where the subscript (i) labels the cases from 1 to m. The gradient ascent is based on

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\sum_{i=1}^{m} \left[h_\theta\left(x^{(i)}\right) - y^{(i)} \right] x_j^{(i)} \tag{3.4}$$

To prevent overfitting for data, one can add L_2 regularization to penalize large magnitude of weights,

$$J(\theta) = \sum_{i=1}^{m} \left[y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1-y^{(i)}\right) \log\left(1-h_\theta\left(x^{(i)}\right)\right) \right] - \frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2, \tag{3.5}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \begin{cases} -\sum_{i=1}^{m} \left[h_\theta \left(x^{(i)} \right) - y^{(i)} \right] x_j^{(i)}, & j = 0 \\ -\sum_{i=1}^{m} \left[h_\theta \left(x^{(i)} \right) - y^{(i)} \right] x_j^{(i)} - \lambda \theta_j, & j \geq 1 \end{cases}, \tag{3.6}$$

where $\lambda > 0$ is the regularization parameter.

3.1.2 Multi-class classification

The logistic regression can be easily generalized to one-vs-all classification problems. Suppose we have K classes in our dataset, the model parameters will be a matrix $\Theta \in R^{K \times (N+1)}$, where each row corresponds to the learned logistic regression parameters for one class. When training the classifier for class $k \in \{1, ..., K\}$, one will want a m-dimensional vector of labels y, where $y_j \in 0, 1$ indicates whether the j-th training instance belongs to class $k\,(y_j = 1)$, or if it belongs to a different class $(y_j = 0)$.

3.1.3 Multi-layer Perceptron

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function $f : R^n \rightarrow R^K$ by training on a dataset, where n is the number of dimensions for input and K is the number of dimensions for output. Given a set of features $X = \{x_1, x_2, ..., x_n\}$ and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from the logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers.

3.1.4 Softmax and cost function

If there are more than two classes, the output function would be a vector of size K(number of classes). Instead of passing through logistic function, it passes through the softmax function in order to keep the summation of predicted probabilities to be equal to 1. The softmax function is defined as

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{K} \exp(z_l)}, \tag{3.7}$$

where z_i represents the i-th element of the input to softmax, which corresponds to class i. The result is a vector containing the probabilities that sample x belong to each class. The output is the class with the highest probability. The cost function for K classes, m training examples and L layer neural network is given by

$$\begin{aligned} J(\theta) &= -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\theta\left(x^{(i)}\right)_k\right) + \left(1 - y_k^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)_k\right)\right] \\ &\quad + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} \left(\theta_{ij}^{(l)}\right)^2, \end{aligned} \tag{3.8}$$

where s_l is the number of nodes in layer l. Notice that the feature label is from 0 to n, so $s_1 = n, s_L = K$, and

$$x^{(i)} = \left(x_0^{(i)}, x_1^{(i)}, \ldots, x_n^{(n)}\right), i = 1, \ldots, m \tag{3.9}$$

with $x_0^{(i)} = 1$.

3.1.5 Activation functions

The choices of activation functions $g(z)$ include

1. Logistic Sigmoid: $g(z) = \frac{1}{1+\exp(-z)}$,

2. Rectified Linear Units (ReLu): $g(z) = \max\{0, z\}$,

3. Hyperbolic Tangent: $g(z) = \tanh(z)$.

As generalized from the logistic regression, the logistic sigmoid function has a gradient easy to compute: $g'(z) = (1 - g(z))g(z)$ and is widely used in feed-forward ANN. ReLu is particularly useful in extracting the non-linear features in the convolutional neural network that will be discussed in the next section.

3.1.6 Backpropagation

The forward propagation for each training example is given by

$$\begin{aligned}
a^{(i)(1)} &= x^{(i)}, \\
z^{(i)(2)} &= \theta^{(1)} a^{(i)(1)}, \\
a^{(i)(2)} &= g\left(z^{(i)(2)}\right), \text{add } a_0^{(i)(2)} \\
&\vdots \\
a^{(i)(L-1)} &= g\left(z^{(i)(L-1)}\right), \\
z^{(i)(L)} &= \theta^{(L-1)} a^{(i)(L-1)}, \\
a^{(i)(L)} &= g\left(z^{(i)(L)}\right).
\end{aligned} \tag{3.10}$$

We could define the error of node j in layer l as $\delta_j^{(i)(l)}$, then by considering the sigmoid activation function,

$$\begin{aligned}
\delta^{(i)(L)} &= a^{(i)(L)} - y^{(i)}, \\
\delta^{(i)(L-1)} &= \left[g\left(z^{(i)(L-1)}\right)\left(1 - g\left(z^{(i)(L-1)}\right)\right) .* \left(\theta^{(L-1)}\right)^T \right]_{s_{L-1}\times K} \delta^{(i)(L)}, \\
&\vdots \\
\delta^{(i)(l+1)} &= \left[g\left(z^{(i)(l+1)}\right)\left(1 - g\left(z^{(i)(l+1)}\right)\right) .* \left(\theta^{(l+1)}\right)^T \right] \delta^{(i)(l+2)} \quad (3.11)
\end{aligned}$$

where s_l is the number of units in layer l. A tedious calculation of gradients gives rise to the result

$$\left(\frac{\partial J(\theta)}{\partial \theta^{(l)}}\right) = \frac{1}{m}\sum_{i'=1}^{m} \delta^{(i')(l+1)} \left(a^{(i')(l)}\right)^T + \frac{\lambda}{m}\theta^{(l)} \quad (3.12)$$

3.2 Convolutional neural network

Convolutional networks, also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology[10]. Examples include time-series data, which can be thought of as 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

3.2.1 The Convolution Operation

The convolution operation on continuous functions is defined as

$$s(t) = (x * w)(t) \equiv \int x(a)w(t-a)da. \tag{3.13}$$

In the discrete version, integration is replaced by summation:

$$s(t) = (x * w)(t) \equiv \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{3.14}$$

The first argument x is referred to as the input and the second argument w is named the kernel. The output is sometimes referred to as the feature map. In the machine learning applications, the input is usually a multi-dimensional array of data and the kernel is usually a multi-dimensional array of parameters that are adapted by the learning algorithm. We will refer to these multidimensional arrays as tensors. Notice that both kernels and input are finite dimensional in practice. For convolution more than 1 axis, we have

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{3.15}$$

It is noticeable that the convolution operation is commutative:

$$s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \tag{3.16}$$

3.2.2 Motivation

Convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivariant representations. Sparse interaction is achieved by making the kernel

smaller than the input. In a deep convolutional neural network, units in the deeper layers may indirectly interact with a larger portion of the input. This allows the network to efficiently describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions. Parameter sharing refers to using the same parameter for more than one function in a model. The particular form of parameter sharing in the case of convolution causes the layer to have a property called equivariance to translation, which means if the input changes, the output changes in the same way. The mathematical definition is given by

$$f(g(x)) = g(f(x)) \tag{3.17}$$

where g can be any function that translates the input.

3.2.3 Pooling

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the max pooling operation reports the maximum output within a rectangular neighborhood. In all cases, pooling helps to make the representation become approximately invariant to small translations of the input.

3.2.4 Full CNN Model

The full architecture (Fig. 3.2) of the CNN model consists of

1. Convolution layer: comprised of feature maps due to multiple feature detectors as Input $\otimes$ Feature detector = Feature Map,

2. ReLu layer: creating nonlinear features by using rectified linear units,

3. Max pooling layer: pooling the maximum in the window,

4. Flattening: converting pooled feature map into 1-d array, which serves as input layer of feedforward neural network,

5. Full connection: feedforward neural network training.

In more sophisticated applications, one can include multiple sets of convolution, ReLU and pooling layers with different number of feature detectors(Fig.3.2).

3.3 Restricted Boltzmann Machine

The graphical model of an RBM[2] is a fully-connected **bipartite** graph as in Fig. 3.3: The nodes are random variables whose states depend on the state of the other nodes they are connected to. The model is therefore parametrized by the weights of the connections, as well as one intercept (bias) term for each visible and hidden unit. The energy function measures the quality of a joint assignment:

$$E(\mathbf{v}, \mathbf{h}) = \sum_i \sum_j w_{ij} v_i h_j + \sum_i b_i v_i + \sum_j c_j h_j. \tag{3.18}$$

In the above formula, $\mathbf{b}$ and $\mathbf{c}$ are the intercept vectors for the visible and hidden layers, respectively. The joint probability of the model is defined in terms of the energy:

$$P(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \tag{3.19}$$

3.3.1 Bernoulli Restricted Boltzmann machines

In the Bernoulli RBM, all units are binary stochastic units. This means that the input data should either be binary, or real-valued between 0 and 1 signifying the probability that the visible unit would turn on or off[11]. This is a good model for character recognition, where the interest is on which pixels are active and which aren't. For images of natural scenes it no longer fits because of background, depth and the tendency of neighbouring pixels to take the same values. In the Bernoulli RBM, both hidden units and visible units are not internally connected while the visible layer and the hidden layer are fully connected to each other. As a result, the conditional probabilities can be factorized as follows:

$$P(\mathbf{v}|\mathbf{h}) = \prod_i P\left(v_i|\,\mathbf{h}\right), \tag{3.20}$$

$$P(\mathbf{h}|\mathbf{v}) = \prod_i P\left(h_i|\,\mathbf{v}\right), \tag{3.21}$$

and

$$P\left(v_i = 1|\mathbf{h}\right) = \text{sigmoid}\left(\sum_j w_{ij}h_j + b_i\right), \tag{3.22}$$

$$P\left(h_i = 1|\mathbf{v}\right) = \text{sigmoid}\left(\sum_i w_{ij}v_i + c_j\right). \tag{3.23}$$

3.3.2 Maximum Likelihood Learning

RBM is an unsupervised learning method, The learning goal is to maximize the logarithmic probability given by

$$\log P(v) = \log \sum_h e^{-E(\mathbf{v},\mathbf{h})} - \log \sum_{x,y} e^{-E(x,y)}. \tag{3.24}$$

To implement gradient descent in minimization, we need the following derivatives of the log-likelihood function:

$$\begin{aligned}\frac{\partial \log P(v)}{\partial w_{ij}} &= -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} + \sum_{\mathbf{h},\mathbf{v}} p(\mathbf{v},\mathbf{h})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} \\ &= P\left(h_i = 1|\mathbf{v}\right) v_j - \sum_{\mathbf{v}} P(\mathbf{v}) P\left(h_i = 1|\mathbf{v}\right) v_j, \end{aligned} \tag{3.25}$$

$$\begin{aligned}\frac{\partial \log P(v)}{\partial b_j} &= -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial b_j} + \sum_{\mathbf{h},\mathbf{v}} p(\mathbf{v},\mathbf{h})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial b_j} \\ &= v_j - \sum_{\mathbf{v}} P(\mathbf{v}) v_j, \end{aligned} \tag{3.26}$$

$$\begin{aligned}\frac{\partial \log P(v)}{\partial c_i} &= -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial c_i} + \sum_{\mathbf{h},\mathbf{v}} p(\mathbf{v},\mathbf{h})\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial c_i} \\ &= P\left(h_i = 1|\mathbf{v}\right) - \sum_{\mathbf{v}} P(\mathbf{v}) P\left(h_i = 1|\mathbf{v}\right), \end{aligned} \tag{3.27}$$

where we have applied

$$\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} = h_i v_j, \tag{3.28}$$

$$\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial b_j} = v_j, \tag{3.29}$$

$$\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial c_j} = h_j. \tag{3.30}$$

3.3.3 Contrastive divergence

Due to the summation over a large number of input data in the gradient evaluation, all common training algorithms for RBMs approximate the log-likelihood gradient given some data and perform gradient ascent on these

approximations. Obtaining unbiased estimates of log-likelihood gradient using MCMC methods typically requires many sampling steps. However, recently it was shown that estimates obtained after running the chain for just a few steps can be sufficient for model training. This leads to contrastive divergence (CD) learning, which has become a standard way to train RBMs.

The idea of k-step contrastive divergence learning (CD-k) is quite simple: Instead of approximating the second term in the log-likelihood gradient by a sample from the RBM-distribution (which would require to run a Markov chain until the stationary distribution is reached), a Gibbs chain is run for only k steps (and usually $k = 1$). As a result,

$$\begin{aligned}
\frac{\partial \log P\left(\mathbf{v}^{(0)}\right)}{\partial w_{ij}} &\approx -\sum_{\mathbf{h}} P\left(\mathbf{h}\left|\mathbf{v}^{(0)}\right.\right) \frac{\partial E\left(\mathbf{v}^{(0)}, \mathbf{h}\right)}{\partial w_{ij}} + \sum_{\mathbf{h}} p\left(\mathbf{h}\left|\mathbf{v}^{(k)}\right.\right) \frac{\partial E\left(\mathbf{v}^{(k)}, \mathbf{h}\right)}{\partial w_{ij}} \\
&= P\left(h_i = 1\left|\mathbf{v}^{(0)}\right.\right) v_j^{(0)} - P\left(h_i = 1\left|\mathbf{v}^{(k)}\right.\right) v_j^{(k)}, \qquad (3.31) \\
\frac{\partial \log P\left(\mathbf{v}^{(0)}\right)}{\partial b_j} &\approx -\sum_{\mathbf{h}} P\left(\mathbf{h}\left|\mathbf{v}^{(0)}\right.\right) \frac{\partial E\left(\mathbf{v}^{(0)}, \mathbf{h}\right)}{\partial b_j} + \sum_{\mathbf{h}} p\left(\mathbf{h}\left|\mathbf{v}^{(k)}\right.\right) \frac{\partial E\left(\mathbf{v}^{(k)}, \mathbf{h}\right)}{\partial b_j} \\
&= v_j^{(0)} - v_j^{(k)}, \qquad (3.32) \\
\frac{\partial \log P\left(\mathbf{v}^{(0)}\right)}{\partial c_j} &\approx -\sum_{\mathbf{h}} P\left(\mathbf{h}\left|\mathbf{v}^{(0)}\right.\right) \frac{\partial E\left(\mathbf{v}^{(0)}, \mathbf{h}\right)}{\partial c_j} + \sum_{\mathbf{h}} p\left(\mathbf{h}\left|\mathbf{v}^{(k)}\right.\right) \frac{\partial E\left(\mathbf{v}^{(k)}, \mathbf{h}\right)}{\partial c_j} \\
&= P\left(h_i = 1\left|\mathbf{v}^{(0)}\right.\right) - P\left(h_i = 1\left|\mathbf{v}^{(k)}\right.\right). \qquad (3.33)
\end{aligned}$$

3.4 Deep auto-encoder

The idea of deep auto-encoder was proposed in the seminal paper by Hinton[11], where stacked RBMs were used to encode image data. The structure of a stacked RBM is shown in Fig.3.4. Instead of a single layer of hidden

units, multiple layers of feature detectors with different dimensions are activated by sigmoid functions except for the top layer. After learning one layer of feature detectors, their activities are treated as input for learning a second layer of features. The first layer of feature detectors then become the visible units for learning the next RBM. This layer-by-layer learning can be repeated as many time as desired. In Hinton's work, the visible units of every RBM had real-valued activities, which were in the range $[0, 1]$ for logistic units. While training higher level RBMs, the visible units were set to the activation probabilities of the hidden units in the previous RBM, but the hidden units of every RBM except the top one had stochastic binary values. The hidden units of the top RBM have stochastic real-valued states drawn from a unit variance Gaussian whose mean was determined by the input from that RBM's logistic visible units. This type of deep auto-encoder had been tested on MNIST data and facial images[11].

It is difficult to optimize the weights in nonlinear auto-encoders that contains multiple layers[11]. With large initial weights, auto-encoders typically find poor local minima; with small initial weights, the gradients in the early layers are tiny, making it infeasible to train auto-encoders with many hidden layers. If the initial weights are close to a good solution, gradient descent in backpropagation works well. This optimal initialization is obtained through a layer-by-layer unsupervised learning algorithm and each layer is trained as in an individual RBM. This has proved to be a very efficient and effective 'pre-training' stage.

In Pankaj's work[14], it was proved and demonstrated that stacked RBM is intimately related to one of the most important and successful techniques in theoretical physics, renormalization group (RG), which is an iterative coarse-graining scheme that allows for the extraction of relevant features as a physical system is examined at different length scales. In their work, an exact mapping between variational RG and deep learning architectures based on RBM was constructed and implemented on the ferromagnetic Ising model. In this report, we followed their work and demonstrated the power of deep auto-encoders using nearest-neighbor Ising Model in two-dimensional square lattice.

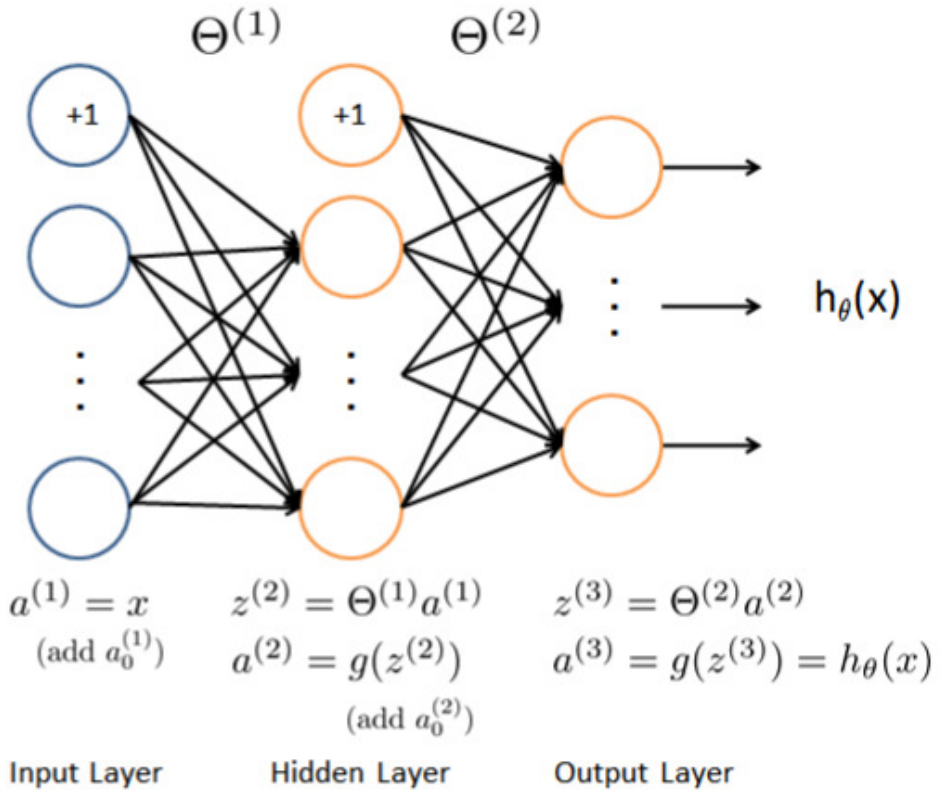


Figure 3.1: The architecture of a feed-forward artificial neural network with a single hidden layer. The output probabilities of each neuron in the hidden layer are served as inputs for the output layer. (This figure was clipped from online course materials at coursera.com)

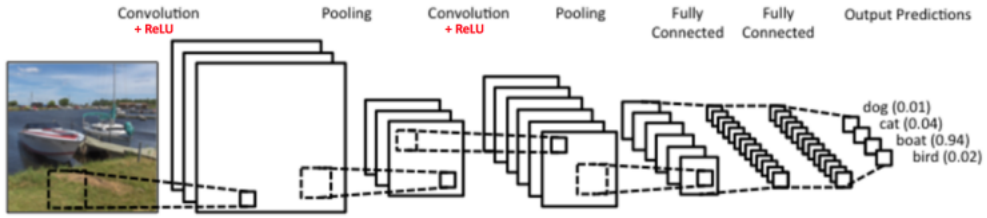


Figure 3.2: The architecture of CNN. The input is a colored picture with a 2d grid of pixels. Two convolution+pooling layers are used to convert pixels into 1d input data for the fully connected layer as in the feed-forward neural network[1].

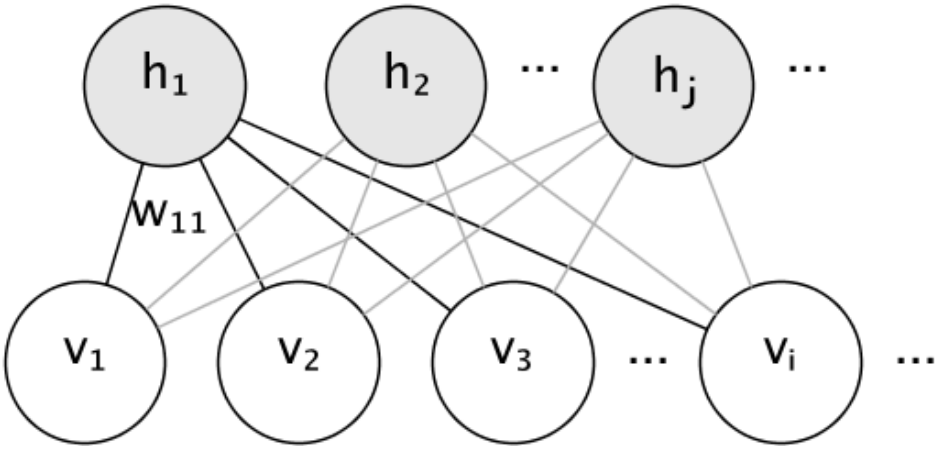
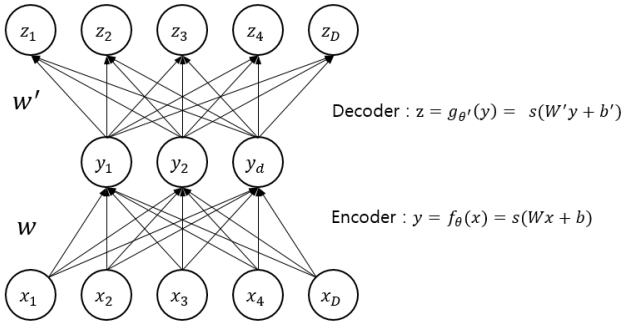


Figure 3.3: The architecture of restricted Boltzmann machine: the hidden units on the top are connected to the visible units at the bottom. There is no internal connections between units in the hidden layer[2].

(a)



(b)

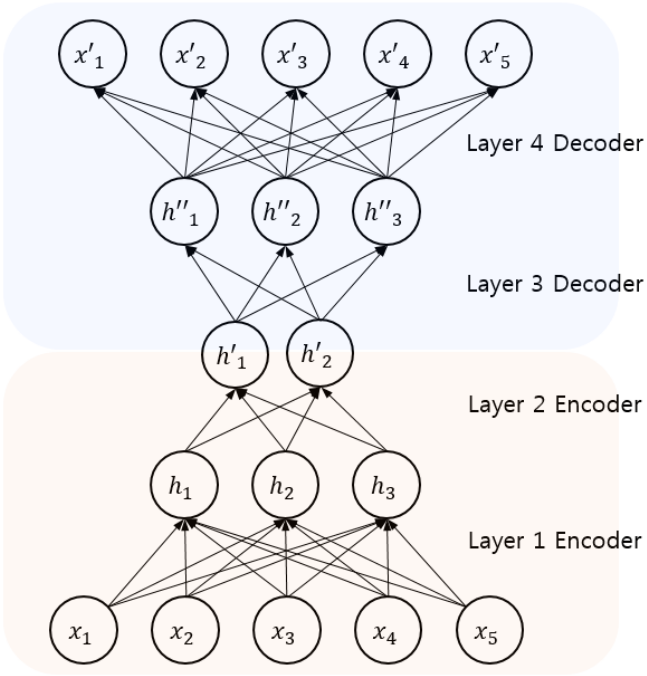


Figure 3.4: The architectures of auto-encoders[3]: (a) Auto-encoder with a single hidden layer and the sigmoid activation function. (b) Deep auto-encoder with stacked RBMs as encoding layers and the decoding layers have the reversed dimensions.

Chapter 4

Data-preparation and model specification

In this chapter, we describe the procedure of data generation, processing and visualization. Then we lay out the structures of our models and specify the choices of parameters.

4.1 Ferromagnetic Ising model

4.1.1 Data

The data for ferromagnetic Ising model was generated by metropolis Markov chain Monte Carlo sampling described in Chapter 2. For training and testing purposes in supervised learning, we generated Ising spin configurations taking binary values $\{1, -1\}$ from 50 different temperatures (k_BT) evenly distributed from 1.0 to 3.6 (We set the Ising coupling constant $J = 1.0$). From each temperature we generated 2500 data points for training sets and 250 data points for test sets. In order to obtain the critical temperature T^* in the thermodynamic limit, we trained our model on datasets of different system size $L = 10, 20, 30, 40, 60$ and the critical temperature for each system size was identified.

For each data point, we labeled the high temperature paramagnetic

phase (PM) as 1 and the low temperature ferromagnetic phase (FM) as 0. The features in the datasets include $L \times L$ spins labeled as $(i_x - 1) \times L + i_y, i_x = 1, 2, \ldots, L; i_y = 1, 2, \ldots, L$. (Table.4.1)

For RBMs and the deep autoencoder in unsupervised learning, we generated Ising spin configurations taking binary values $\{0, 1\}$ from system size $L = 40$ with temperature $k_B T = 2.451$ (close to the critical temperature). Our sample size was 50000 and we applied a train-test split with test size proportion equal to 0.2 in the whole dataset.

temperature	Spin1	Spin2	$\cdots$	Spin400	phase
3.548	1	-1	$\cdots$	1	1
3.548	-1	1	$\cdots$	-1	1
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Table 4.1: Sample data frame for $L = 20$ Ising model.

4.1.2 Model

For supervised learning, we used a feed-forward ANN with one hidden layer containing 400 sigmoid hidden units to classify the binary phases in the Ising model. The optimizer was chosen to be stochastic gradient descent (SGD) with batch size equal to 100 the number of epochs equal to 10. Our learning rate was 0.001 and the weight decay was equal to $1e - 6$.

For unsupervised learning, we implemented a deep auto-encoder with stacked RBMs as illustrated in Fig.3.4. The visible layer of the first RBM

contains $40 \times 40 = 1600$ binary units and the first hidden layer contains 400 units taking real values within $[0, 1)$ serving as the visible units for the second RBM. The third layer contain 100 units and our encoding model was topped with 25 hidden units. Our encoding model was trained layer by layer with contrastive divergence for 200 epochs and momentum 0.5 using batch size 100 on 40000 total training samples.

4.2 Ising gauge model

4.2.1 Data

The data for Ising gauge model was generated by metropolis Markov chain Monte Carlo sampling described in Chapter 2 with both spin flips and gauge updates where we chose $s = 0.75$. For training and testing purposes in supervised learning, we generated Ising gauge spin configurations taking binary values $\{1, -1\}$ from system size $L = 4, 8, 12, 16, 20, 24, 28$ with T close to 0 and ∞ respectively (Again we set the Ising coupling constant $J = 1.0$). From each temperature we generated 25000 data points for training sets and 2500 data points for test sets.

For each data point, we labeled the high temperature deconfined phase as 0 and the low temperature confined phase as 1. The features in the datasets include $2 \times L \times L$ spins defined on the links. We divided our spins into two sublattices: spins on x links and spins on y links, Spins on x links are labeled as $(i_x - 1) \times L + i_y, i_x = 1, 2, \ldots, L; i_y = 1, 2, \ldots, L$, while spins on y links are labeled as $L \times L + (i_x - 1) \times L + i_y, i_x = 1, 2, \ldots, L; i_y = 1, 2, \ldots, L$.

As input for CNN model, one needs to reshape the data into $(2, L, L)$ such that two-dimensional feature detectors can be applied to scan the lattice.

4.2.2 Model

The model we implemented includes a convolutional layer first followed by a ReLu layer, and then followed by a 2-d max pooling layer. The convolutional layer applies 64 2×2 filters to the configuration on each sublattice and the pooled features were then flattened into 1-d arrays. The outcome was served as input data into a fully connected layer with 64 units and a softmax output layer. To prevent overfitting, a dropout regularization rate $p = 0.1$ was added to the fully connected layer.

Chapter 5

Results

5.1 Feed-forward ANN on ferromagnetic Ising model

As mentioned in the previous chapter, our feed-forward ANN was trained on 50 different temperatures with 2500 data points per temperature. The predicted probabilities for the test set were averaged over 250 data points per temperature and shown in Fig.5.1. The transition temperature from FM to PM (i.e. critical temperature) was decided by the intersection between the output layer and the base model $p = 0.5$. We noticed that with the increase of system size L, the output probability drops (increases) more sharply around the critical temperature. This is due to $O(L^2)$ increase in the number of input features. The training and test accuracy for different L's are listed in Table.5.1.

L	Training accuracy	Test accuracy
10	0.9321	0.9306
20	0.9721	0.9636
30	0.9809	0.9834
40	0.9762	0.9844
60	0.9944	0.9934

Table 5.1: Training and test accuracy for ferromagnetic Ising model.

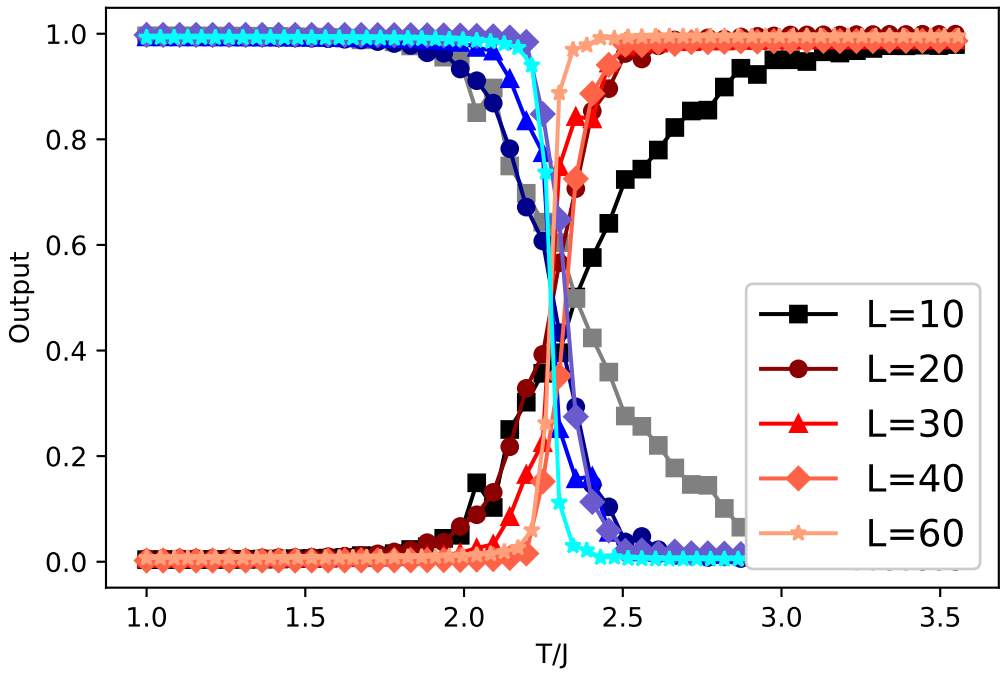


Figure 5.1: The output layer of our trained feed-forward neural network model averaged over a test set as a function of T/J for the square-lattice ferromagnetic Ising model of size $L = 10, 20, 30, 40, 60$.

From the predicted critical temperature T_c for $L = 10, 20, 30, 40, 60$, we could estimate the critical temperature T^* in the thermodynamic limit $L = \infty$ by finite size scaling. The procedure was carried out first by a linear regression $y = w * x + b$ with the least square fitting, where the response variable y is T_c and the input variable is $1/L$, and then by extracting the intercept b as an estimate of T^*. The T-$1/L$ plot and the fitted line are shown in Fig. 5.2 and the scaling analysis estimates $T^*/J \approx 2.266$, which is pretty close to the

theoretical value $T^*/J = 2/ln(1+\sqrt{2}) \approx 2.269$.

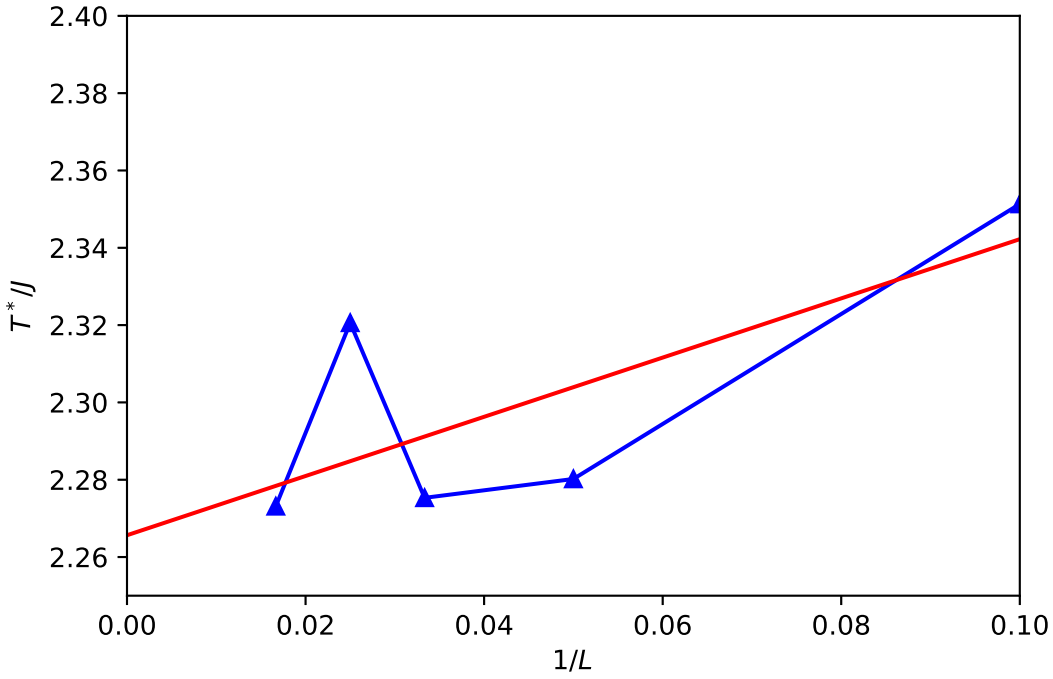
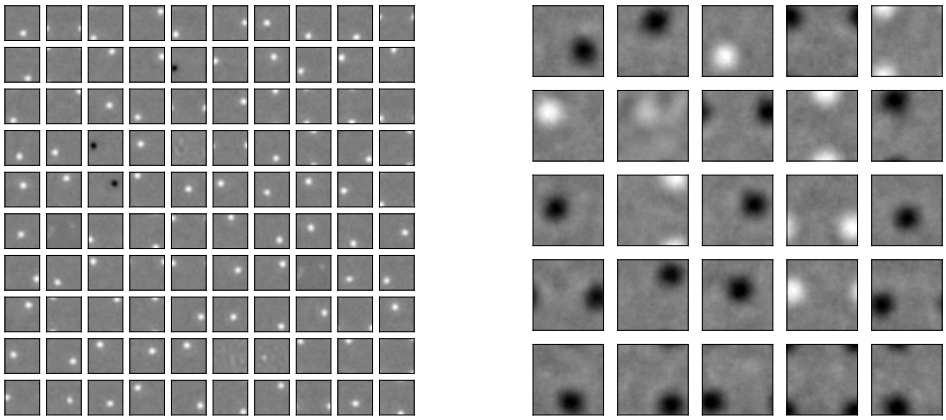


Figure 5.2: The plot for finite size scaling of the critical temperature $T_c/J \sim 1/L$. The red line is the fitted result.

5.2 Deep auto-encoder on ferromagnetic Ising model

As mentioned in Chapter 3, our deep auto-encoder consists of stacked RBMs with 400-100-25 hidden units. We performed a layer-by-layer pretraining on the encoding layers. After the weights and biases for each RBM were obtained, the model was 'unfolded' to produce encoder and decoder networks.



(a) (b)

Figure 5.3: Visualization of effective receptive fields for (a) 100 spins in the middle layer, (b) 25 spins in the top layer. Each 40 by 40 pixel image depicts the effective receptive field of one of the spins in the hidden units.

Since the decoded layer has reversed dimensions between visible units and hidden units compared to the encoded layer, the weight matrix for the decoded layer is the transpose of the one in the encoded layer and the biases in the visible units and the hidden units are switched between the two layers. With this initialization of parameters, one can perform back-propagation in the global fine-tuning stage with a much better starting point compared to random initialization.

To better understand the unsupervised learning results in the pretraining stage, the effective receptive fields[14] are defined as follows: We denote the effective receptive field matrix of layer l by $r^{(l)}$ and the number of spins in layer l by $n^{(l)}$, with the visible layer corresponding to $l = 0$. Each column in

$r^{(l)}$ is a vector that encodes the receptive field of a single spin in hidden layer l. It can be computed by convoluting the weight matrices $W^{(l)}$ encoding the weights w_{ij} between the spins in layers $l-1$ and l. To compute $r^{(l)}$ first we set $r^{(1)} = W^{(1)}$ and used the recursion relationship $r^{(l)} = r^{(l-1)}W^{(l)}$ for $l > 1$. Thus the effective field is a measure of how much that hidden spin influences the spins in the visible layer and a way to visualize which spins in the visible layer that coupled to that hidden spin. In Fig.5.3, we plotted the density of the effective receptive fields by color maps for the middle layer with 100 spins and for the top layer with 25 spins. It can be observed that the receptive fields are localized in the visible spins and get larger as one moves up the in the encoded layers. As pointed out in Pankaj's work[14], this is consistent with what is expected from the successive application of block renormalization group method in statistical mechanics, indicating that the architecture of stacked RBMs is implementing a coarse-graining scheme similar to block spin renormalization. Moreover, the size of the blocks coupling to each hidden unit in a layer are of approximately the same size. In conclusion, this local block spin structure emerging from the pretraining process strongly suggests that stacked RBMs are self-organizing to implement block-spin renormalization.

In the back-propagation fine-tuning stage, we compared the training result with that of random initialization in Fig.5.4, where the logarithmic likelihood loss as a function of epochs was monitored for randomly initialized and pre-trained models. By observation, one is able to find that the deep auto-encoder makes rapid progress after pre-training but little progress without

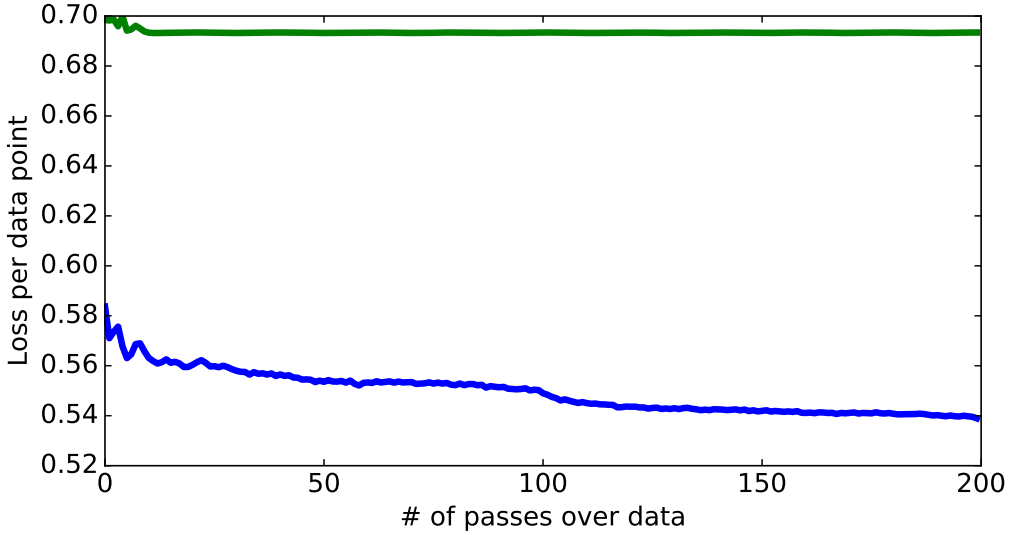


Figure 5.4: Loss per data point in the fine tuning stage as a function of number of epochs trained for the deep auto-encoder. The green curve corresponds to random initialization in the model parameter. The blue curve is initialized with pre-trained model parameters.

pre-training. This is consistent with the result from Hinton's work[11].

With the fine-tuned parameters, our model can be applied to reconstruct the Ising spin configurations in the test dataset. In Fig.5.5, we generated the reconstructed spin configurations by Bernoulli sampling based on the predicted probabilities for each spin on the lattice sites.

5.3 CNN on Ising gauge model

We employed CNN to train the Ising gauge models with system size $L = 4, 8, 12, 16, 20, 24, 28$. The CNN discriminates high-temperature $T = \infty$

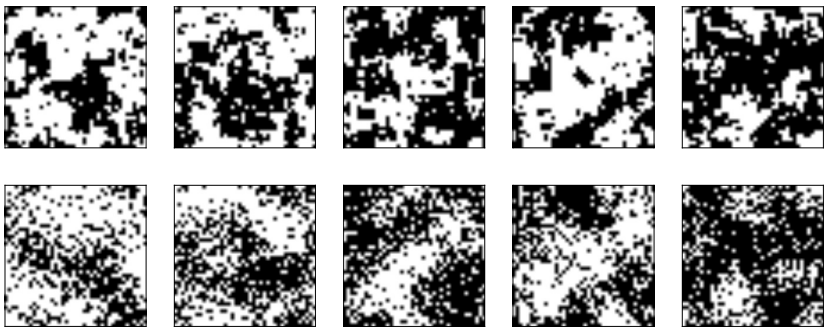


Figure 5.5: Reconstructed spin configurations (lower) compared to the corresponding raw configurations (upper).

from $T = 0$ states with an accuracy of 100% on both training and test datasets except for $L = 4$, in which 95.54% and 95.15% accuracy was achieved on the training and test set respectively. To examine the origin of the discriminative power of CNN, one can check whether it discerns $T = 0$ states from $T = \infty$ states by the presence and absence of satisfied local constraints, i.e. whether the 4-spin state on a plaquette q is such that $\prod_i \sigma_i^z = 1$. We constructed new test sets with modified low temperature states: applying transformations where a spin is flipped every $m = 2$ and $m = 4$ plaquettes (Fig. 5.6). Such transformations were proved not to destroy the topological order of the $T = 0$ state where the extended closed loop structure was preserved, but they change the local constraints of the original Ising lattice gauge theory. Our model for $L = 20$ showed that the prediction accuracy drops to 50% and 82.77% for $m = 2$ and $m = 4$ respectively. This reveals that CNN relies on the presence

of local constraints instead of the extended loop structure.

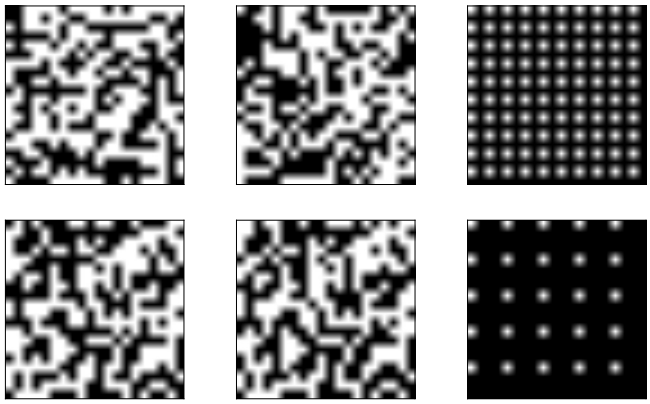


Figure 5.6: The spin flip transformations on the links in x direction every m plaquettes for $L = 20$, where $m = 2$ for the upper row and $m = 4$ for the lower row. From left to right: spin configurations before the transformation, spin configurations after the transformation, spins changed under the transformation indicated by the white spots.

Chapter 6

Conclusion and discussion

In this report, we presented our study on the deep neural network application in the Ising type models on the square lattice. Our results mostly reproduced the work by Juan[7] and Pankaj[14] and showed the discriminative and encoding power of deep neural networks.

In the application of feed-forward ANN to the ferromagnetic Ising model, one can understand the training of the network through a simple toy model involving a hidden layer of only three analytically 'trained' perceptrons[7], representing the possible combinations of high- and low-temperature magnetic states exclusively on the basis of their magnetization, which plays the role of the order parameter in the model. The fully connected ANN relies on the order parameter in the classification task. It has also been shown that the power of feed-forward ANNs lies in their ability to generalize to different lattice structures[7].

In the application of CNN to Ising gauge model, we have shown that deep learning methods can encode basic information about unconventional phases and we demonstrated the power of CNN in local constraints detection even though the model itself does not have an order parameter. In Juan[7]'s

work, a toy model using streamlined version of the original CNN was constructed to detect satisfied local constraints and estimate the cross over temperature. The analytical model they presented indicated that neural networks have the potential to represent ground state wavefunctions of unconventional topological states.

In the application of deep auto-encoders, we illustrated by Ising model that the mechanism behind stacked RBMs is similar to the iterative coarse-graining procedure in RG. We also demonstrated the influence of pre-training in reconstructing the spin configurations in analogy to the image and the hand-written digit reconstruction problems.

Overall, our study suggests a prospectively enormous applications of deep learning algorithms in solving notoriously difficult many-body problems in physics. As in all other areas of "big data", we anticipate the rapid adoption of machine learning techniques as a basic research tool in statistical mechanics.

Appendix

Appendix 1

Sample code

We present sample codes in Python here for both supervised and unsupervised learning.

1.1 Keras implementation of feed-forward ANN in ferromagnetic Ising model

```
# coding: utf-8

# In[1]:

import pandas as pd
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras import regularizers
import matplotlib.pyplot as plt
get_ipython().magic(u'matplotlib inline')


# In[2]:

def read_data(L):
    """
    :type L: int(system size)
```

```
    :rtype train, test: pd.dataframe(with phases labeled)
    """
    train_filename='configuration'+str(L)+'.csv'
    test_filename='test'+str(L)+'.csv'
    columns=['temperature']+['Spin'+str(i) for i in
        xrange(1,L*L+1)]
    train=pd.read_csv('Ising_data/'+'L'+str(L)+'/'+
        train_filename,names=columns)
    test=pd.read_csv('Ising_data/'+'L'+str(L)+'/'+
        test_filename,names=columns)

    # decide critical temperature
    filename='Cv'+str(L)+'.csv'
    specific_heat=pd.read_csv('Ising_data/'+'L'+str(L)+'
        /'+filename,names=['temperature','Cv'])
    Tc=specific_heat['temperature'][np.argmax(
        specific_heat['Cv'])]

    # add phase column
    train['FM']=[int(T<=Tc) for T in train['temperature'
        ]]
    test['FM']=[int(T<=Tc) for T in test['temperature']]
    train['PM']=[int(T>Tc) for T in train['temperature'
        ]]
    test['PM']=[int(T>Tc) for T in test['temperature']]

    return train, test


# In[3]:

train, test=read_data(20)


# In[25]:

train.tail()
```

```
# In[4]:

def data_process(train, test):
    """
    :type train,test: pd.dataframe(with phases labeled)
    :rtype trX, trY, teX, teY
    """
    # shuffle
    train=train.sample(frac=1).reset_index(drop=True)

    trX, trY = train.drop(['FM','PM','temperature'],axis
        =1), np.array(train[['FM','PM']])
    teX, teY = test.drop(['FM','PM','temperature'],axis
        =1), np.array(test[['FM','PM']])

    return trX, trY, teX, teY


# In[5]:

trX, trY, teX, teY=data_process(train, test)


# In[18]:

def create_model(hidden_units=3, lamb=0.05,lr=0.001):
    classifier=Sequential()
    classifier.add(Dense(output_dim=hidden_units, init='
        uniform', kernel_regularizer=regularizers.l2(lamb
        ),                          activation='sigmoid',
        input_dim=trX.shape[1]))
    classifier.add(Dense(output_dim=1, init='uniform',
        kernel_regularizer=regularizers.l2(lamb),
                             activation='sigmoid'))
```

```
    sgd = SGD(lr=0.001, decay=1e-6)#, momentum=0.9,
        nesterov=True)
    classifier.compile(optimizer='adam',loss='
        binary_crossentropy',metrics=['accuracy'])
    return classifier


# In[19]:

classifier=create_model(hidden_units=400)


# In[20]:

classifier.fit(np.array(trX),np.argmax(trY,axis=1),
    batch_size=100, nb_epoch=5,verbose=1)


# In[21]:

y_pred=classifier.predict(np.array(teX))


# In[22]:

np.mean((y_pred>0.5)[:,0]==np.argmax(teY,axis=1))


# In[23]:

Temperatures=np.array(sorted(list(set(test['temperature'
    ]))))
Pred_avgs=[]
num_T=Temperatures.shape[0]
num_test=len(test['temperature'])
batch=num_test/num_T
```

```
for i in xrange(num_T):
    start=batch*i
    end=start+batch
    batch_prediction=y_pred[start:end]
    Pred_avgs.append(np.mean(batch_prediction,axis=0))

Pred_avgs=np.array(Pred_avgs)


# In[24]:

import matplotlib.pyplot as plt
get_ipython().magic(u'matplotlib inline')

plt.plot(Temperatures, Pred_avgs[:,0], 'bo',
   Temperatures, Pred_avgs[:,0], 'b-',
   Temperatures, 1-Pred_avgs[:,0], 'ro', Temperatures,
   1-Pred_avgs[:,0], 'r-')


# In[ ]:
```

1.2 Keras implementation of CNN in Ising gauge model

```
# coding: utf-8

# In[1]:

import pandas as pd
import numpy as np


# In[2]:

def read_data(L):
    """
    :type L: int(system size)
    :rtype train,test: pd.dataframe(with phases labeled)
```

```
    """
    train_filename='configuration'+str(L)+'.csv'
    test_filename='test'+str(L)+'.csv'
    columns=['temperature']+['Spin'+str(i) for i in
        xrange(1,2*L*L+1)]
    train=pd.read_csv('Ising_gauge_data/'+'L'+str(L)+'/'
        +train_filename,names=columns)
    test=pd.read_csv('Ising_gauge_data/'+'L'+str(L)+'/'+
        test_filename,names=columns)

    # Separate T=0 and T=inf
    Tc=2.

    # add phase column (T=0: 1, T=inf: 0)
    train['phase']=[int(T<=Tc) for T in train['
        temperature']]
    test['phase']=[int(T<=Tc) for T in test['temperature
        ']]

    return train, test


# In[3]:

L=8
train, test=read_data(L)


# In[4]:

train.head()


# In[5]:

def data_process(train, shuffle=True):
    """
```

```
    :type train, test: pd.dataframe(with phases labeled)
    :rtype trX, trY, teX, teY
    """
    # shuffle
    if shuffle:
        train=train.sample(frac=1).reset_index(drop=True
            )

    # separate A, B sublattices
    trainA=train[['Spin'+str(i) for i in xrange(1,L*L+1)
        ]].as_matrix()
    trainB=train[['Spin'+str(i) for i in xrange(L*L+1,2*
        L*L+1)]].as_matrix()
    trX=np.zeros((trainA.shape[0], L, L, 2))
    trX[:,:,:,0]=np.reshape(trainA,(trainA.shape[0], L,
        L))
    trX[:,:,:,1]=np.reshape(trainB,(trainB.shape[0], L,
        L))

    trY = train['phase']

    return trX, trY


# In[6]:

trX, trY=data_process(train)
teX, teY=data_process(test)


# **check plaquette**

# In[7]:

def plaquette(links):
    plaquette = np.zeros((links.shape[0], L, L))
```

```
    for i in xrange(L):
        for j in xrange(L):
            plaquette[:, i, j]=links[:, i, j, 0]*links
               [:, i, j, 1]              *links[:, (i+1)%L
               , j, 1]*links[:, i, (j+1)%L, 0]

    return plaquette


# In[8]:

plaq=plaquette(trX)
plaq


# In[9]:

# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from keras.optimizers import Adam


# In[10]:

# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Conv2D(64, (2, 2), input_shape = (L, L,
   2), activation = 'relu'))

# Step 2 - Pooling
```

```
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 64, activation = 'relu'))
# drop out
classifier.add(Dropout(p = 0.1))

classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
adam=Adam(lr=0.0001)
classifier.compile(optimizer = adam, loss = '
    binary_crossentropy', metrics = ['accuracy'])


# In[11]:

classifier.fit(trX,trY,batch_size=100, nb_epoch=25,
    verbose=0)


# In[12]:

try_pred=classifier.predict(trX)
tey_pred=classifier.predict(teX)


# In[13]:

np.mean((try_pred>0.5)[:,0]==trY)


# In[14]:
```

```
tey_pred.shape


# In[15]:

np.mean((tey_pred>0.5)[:,0]==teY)


# **predict cross-over temperature on finite temperature
    data**:

# In[16]:

def read_finite(L):
    filename='test_finiteT'+str(L)+'.csv'
    columns=['beta']+['Spin'+str(i) for i in xrange(1,2*
        L*L+1)]
    data=pd.read_csv('Ising_gauge_data/'+'L'+str(L)+'/'+
        filename,names=columns)

    return data


# In[17]:

def data_process_finite(test):
    """
    :type train,test: pd.dataframe(with phases labeled)
    :rtype trX, trY, teX, teY
    """
    # separate A, B sublattices
    testA=test[['Spin'+str(i) for i in xrange(1,L*L+1)
        ]].as_matrix()
    testB=test[['Spin'+str(i) for i in xrange(L*L+1,2*L*
        L+1)]].as_matrix()
    teX=np.zeros((testA.shape[0], L, L, 2))
    teX[:,:,:,0]=np.reshape(testA,(testA.shape[0], L, L)
```

```
        )
    teX[:,:,:,1]=np.reshape(testB,(testB.shape[0], L, L)
        )

    return teX


# In[18]:

test_finite=read_finite(L)
teX=data_process_finite(test_finite)


# In[19]:

y_pred=classifier.predict(teX)


# In[20]:

y_pred=(y_pred>0.5)


# In[21]:

Betas=np.array(sorted(list(set(test_finite['beta']))))
Pred_avgs=[]
num_T=Betas.shape[0]
num_test=len(test_finite['beta'])
batch=num_test/num_T

for i in xrange(num_T):
    start=batch*i
    end=start+batch
    batch_prediction=y_pred[start:end]
    Pred_avgs.append(np.mean(batch_prediction,axis=0))
```

```
Pred_avgs=np.array(Pred_avgs)


# In[22]:

import matplotlib.pyplot as plt
get_ipython().magic(u'matplotlib inline')

plt.plot(Betas, Pred_avgs[:,0], 'bo', Betas, Pred_avgs
    [:,0], 'b-',              Betas, 1-Pred_avgs[:,0], 'ro',
    Betas, 1-Pred_avgs[:,0], 'r-')


# In[23]:

teX.shape


# In[24]:

batch


# In[ ]:
```

Bibliography

[1] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[2] http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.BernoulliRBM.html.

[3] https://wikidocs.net/3413.

[4] https://en.wikipedia.org/wiki/Statistical_physics, Jul 2017.

[5] https://en.wikipedia.org/wiki/Machine_learning, Jul 2017.

[6] Louis-Fran çois Arsenault, Alejandro Lopez-Bezanilla, O. Anatole von Lilienfeld, and Andrew J. Millis. Machine learning for many-body physics: The case of the anderson impurity model. *Phys. Rev. B*, 90:155136, Oct 2014.

[7] Juan Carrasquilla and Roger G. Melko. Machine learning phases of matter. *Nat Phys*, 13(5):431–434, May 2017. Letter.

[8] Wen Xiao Gang. *Quantum field theory of many-body systems: from the origin of sound to an origin of light and electrons*. Oxford University Press, Oxford, 2007.

[9] Luca M. Ghiringhelli, Jan Vybiral, Sergey V. Levchenko, Claudia Draxl, and Matthias Scheffler. Big data of materials science: Critical role of the descriptor. *Phys. Rev. Lett.*, 114:105503, Mar 2015.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. *MIT press*, page 1, 2016.

[11] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[12] Sergei V. Kalinin, Bobby G. Sumpter, and Richard K. Archibald. Big-deep-smart data in imaging for guiding materials design. *Nat Mater*, 14(10):973–980, Oct 2015. Progress Article.

[13] Aaron Gilad Kusne, Tieren Gao, Apurva Mehta, Liqin Ke, Manh Cuong Nguyen, Kai-Ming Ho, Vladimir Antropov, Cai-Zhuang Wang, Matthew J. Kramer, Christian Long, and Ichiro Takeuchi. On-the-fly machine-learning for high-throughput experiments: search for rare-earth-free permanent magnets. *Sci. Rep.*, 4:6367, Sep 2014.

[14] Pankaj Mehta and David J. Schwab. An exact mapping between the Variational Renormalization Group and Deep Learning. *arXiv*, pages 1–7, oct 2014.

[15] Anders W. Sandvik. Computational studies of quantum spin systems. *AIP Conference Proceedings*, 1297(1):135–338, 2010.

[16] S. S. Schoenholz, E. D. Cubuk, D. M. Sussman, E. Kaxiras, and A. J. Liu. A structural approach to relaxation in glassy liquids. *Nat Phys*, 12(5):469–471, May 2016. Letter.

[17] J. Thijssen. *Computational Physics*. Cambridge University Press, 2007.

[18] S Wansleben and A Weinkauf. Monte carlo study of an ising gauge system. *Z. Phys. B*, 261:255–261, 1983.

Vita

Qi Chen graduated from Kongjiang Senior High School in Shanghai, China. He received the Batchelor of Science in Applied Physics from Tongji University in 2011. Then he started his Ph.D in Physics at the University of Texas at Austin from August 2011. In May 2015, he was enrolled as a M.S. student in statistics and data science at the University of Texas at Austin.

Address: chenqi0805@gmail.com

This report was typeset using LaTeX by the author.